

Gyakorlat

MFSPRING1

1. Feladat

Készítsünk el egy banki átutalást megvalósító alkalmazást, egyelőre Spring keretrendszer használata nélkül.

Adatbázis szerkezet:

Tábla név: **ACCOUNTS**

Oszlopok: **ACCOUNT_NAME VARCHAR(100) UNIQUE,**
ACCOUNT_BALANCE INT NOT NULL

Az alkalmazás központi eleme egy `TransferService` interface-t megvalósító osztály a `TransferServiceImpl` legyen, melyben valósítsuk meg az alábbi metódust:

```
public Confirmation transfer(String fromAccountName, String toAccountName, int amount)
```

ahol

- `fromAccountName`: a terhelendő számla neve
- `toAccountName`: a cél számla neve
- `amount`: átutalandó összeg

A `TransferService` interface alatt az alábbi metódussal tudjuk beállítani, a konkrét `AccountRepository` megoldást:

```
public void setAccountRepository(AccountRepository accountRep);
```

A megvalósítás során először olvassuk ki az adott `from` és `to account`-okat adatbázisból. Ha még nem léteznek, akkor 10000-es kezdő egyenleggel szúrjuk be őket. Ezután végezzük el az átutalást (`fromAccount` egyenleg csökkentése, `toAccount` egyenleg növelése) és `update`-eljük az adatbázis rekordokat. Az átutalás végén készítsünk nyugtát (`Confirmation` osztály), amelyet adjunk vissza a hívónak.

Továbbá hozzunk létre egy `AccountRepository` interface-t megvalósító osztályt `JdbcAccountRepository` néven, amely az alábbi metódusokat valósítja meg:

- ```
public Account loadAccount(String accountNo) throws SQLException;
```
- ```
public void updateAccount(Account account) throws SQLException;
```
- ```
public void insertAccount(Account account) throws SQLException;
```

Entitások/adatszerkezetek:

Account

```
private String accountName: account neve
```

```
private int balance: számlaegyenleg
```

Confirmation

```
private int newBalance: fromAccount számla új egyenlege
```

```
private boolean success: true/false, sikeres/sikertelen átutalás
```

```
private String message: formázott üzenet
```

### **Megoldás**

VS Code indítása után hozzunk létre egy új Maven projectet. Ctrl+Shift+P billentyűvel futtassuk a „Spring Initializr: Create a Maven Project” parancsot az alábbi beállításokkal:

Spring Boot version=3.1.2

Project language=Java

Group Id=hu.masterfield

Artifact Id=bankproject

Packaging type=Jar

Java version=17

Dependencies:

- Spring Boot DevTools
- JDBC API
- Spring Data JPA
- Maria DB Driver
- Thymeleaf
- Spring Web
- Jersey

Masterfield Oktatóközpont  
Spring programozás tanfolyam  
2024. november 13.  
Félegyházi Dávid

A létrejött projekt struktúra elemei:

Az alábbi Java csomag struktúrát fogjuk kialakítani:

hu.masterfield.bankproject.datatypes: adattároló osztályok

hu.masterfield.bankproject.interfaces: interface-ek

hu.masterfield.bankproject.services: interface-eket megvalósító osztályok

hu.masterfield.bankproject.exceptions: saját exception-ök

hu.masterfield.bankproject.test: teszt osztályok

Hozzuk létre az alábbi interface-eket a hu.masterfield.bankproject.interfaces alá:

AccountRepository

TransferService

Hozzuk létre az adatszerkezeteket a hu.masterfield.bankproject.datatypes alá:

Confirmation

Account

Hozzuk létre a JdbcAccountRepository és a TransferServiceImpl osztályt a hu.masterfield.bankproject.services alá. Valósítsuk meg a metódusokat. Az adatbázis kapcsolat felépítését a JdbcAccountRepository-ban valósítsuk meg:

```
public class JdbcAccountRepository implements AccountRepository {
 private static String dbURL =
 "jdbc:mariadb://localhost:3306/transfer";

 static Connection createConnection() {
```

```
Connection conn = null;
try {

 Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
 conn = DriverManager.getConnection(dbURL, "root", "root");

} catch (Exception except) {
 except.printStackTrace();
}
return conn;
}

private static Connection connection = createConnection();

public JdbcAccountRepository() {}
```

Indítsuk el a MariaDB adatbázist (Windows service).

Hozzuk létre a **transfer** adatbázist MariaDB alatt:

```
c:\MariaDB 10.6\bin>mariadb -u root -p
Enter password: ****
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.6.11-MariaDB mariadb.org binary
distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab
and others.

Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.

MariaDB [(none)]> create database transfer;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]>
```

VS Code alatt vegyünk fel egy új SQL kapcsolatot: Ctrl+Shift+P > SQLTools Management: Add New Connection...

Válasszuk a MariaDB-t és adjuk meg a kapcsolat beállításait:

Masterfield Oktatóközpont  
Spring programozás tanfolyam  
2024. november 13.  
Félegyházi Dávid

Teszteljük és siker esetén mentsük a kapcsolatot.

Csatlakozunk a kapcsolathoz. A létrejött transfer.session.sql file-ból hozzuk létre az adatbázis táblát az alábbi sql-lel:

```
create table transfer.ACCOUNTS (
 ACCOUNT_NAME VARCHAR(100) UNIQUE,
 ACCOUNT_BALANCE INT NOT NULL
)
```

Hozunk létre egy teszt osztályt a hu.masterfield.bankproject.test alá TransferTest néven.

Valósítsuk meg a main metódust:

```
public static void main(String[] args) {
 AccountRepository accRep = new JdbcAccountRepository();
 TransferService serv = new TransferServiceImpl(accRep);

 Confirmation conf = serv.transfer("Szegény Sándor", "Pénzes Zsolt", 999);
 System.out.println(conf);
}
```

## 2. Feladat

Módosítsuk a banki átutalást megvalósító alkalmazásunkat, hogy a Spring keretrendszeren belül fusson. Definiáljunk Spring konfigurációt, bean-eket (TransferService, AccountRepository, Connection). Emeljük ki az adatbázis kapcsolat paramétereit a Spring konfigurációba.

### Megoldás

Hozzuk létre a hu.masterfield.bankproject.configuration csomag alatt az ApplicationConfiguration osztályt. Valósítsuk meg a bean-eket.

```
@Configuration
public class ApplicationConfiguration {

 private static String dbURL =
 "jdbc:mariadb://localhost:3306/transfer";

 @Bean
 public TransferService transferService() {
 return new TransferServiceImpl(accountRepository());
 }

 @Bean
 public AccountRepository accountRepository() {
 return new JdbcAccountRepository(connection());
 }

 @Bean
 public Connection connection() {
 return createConnection();
 }

 private Connection createConnection() {
 Connection conn = null;
 try {

 Class.forName("org.mariadb.jdbc.Driver").newInstance();
 conn = DriverManager.getConnection(dbURL, "root", "root");

 } catch (Exception except) {
 except.printStackTrace();
 }
 return conn;
 }
}
```

A JdbcAccountRepository osztályból távolítsuk el a kapcsolat létrehozásával összefüggő részeket.

Hozzuk létre a Spring indító teszt osztályt TransferTestSpring néven:

```
public static void main(String[] args) {
 ApplicationContext context =
 SpringApplication.run(ApplicationConfiguration.class);

 TransferService service = (TransferService) context.getBean("transferService");

 Confirmation conf = service.transfer("Szegény Sándor", "Pénzes Zsolt", 999);
 System.out.println(conf);
}
```

Az application.properties-ben tiltsuk le a webes konténert:

```
spring.main.web-application-type=none
```

Futtassuk a teszt osztályunkat.

Masterfield Oktatóközpont  
Spring programozás tanfolyam  
2024. november 13.  
Félegyházi Dávid

### 3. Feladat

Készítsünk JUnit teszt osztályt az átutalás tesztelésére, TransferServiceTests néven.

#### Megoldás

Adjunk hozzá egy új JUnit Test Case-t a hu.masterfield.bankproject csomaghoz

TransferServiceTests néven:

```
public class TransferServiceTests {
 private TransferService service;

 @BeforeEach
 public void setUp() {
 ApplicationContext context =
 SpringApplication.run(ApplicationConfiguration.class);
 service = context.getBean(TransferService.class);
 }

 @Test
 public void transfer() {
 Confirmation receipt;
 receipt = service.transfer("Szegény Sándor", "Pénzes Zsolt", 999);
 System.out.println(receipt.getMessage());
 assertEquals(10000 - 999, receipt.getNewBalance());
 }
}
```

Masterfield Oktatási Központ  
Spring programozás tanfolyam  
2024. november 1. napján  
Félegyházi Dávid



#### 4. Feladat

Válasszuk szét az ApplicationConfiguration-t: InfraConfiguration + ApplicationConfig konfigurációs osztályokra. Az InfraConfiguration tartalmazza az adatbázis eléréshez szükséges kapcsolat létrehozását.

A Connection definiálásánál használjuk az @Autowired annotációt.

#### Megoldás

Hozzuk létre az InfraConfiguration osztályt a hu.masterfield.bankproject.configuration csomag alá. Jelöljük @Configuration annotációval.

@Configuration

```
public class InfraConfiguration {

 private static String dbURL =
 "jdbc:mariadb://localhost:3306/transfer";

 @Bean
 public Connection connection() {
 return createConnection();
 }

 private Connection createConnection() {
 Connection conn = null;
 try {
 Class.forName("org.apache.derby.jdbc.ClientDriver");
 conn = DriverManager.getConnection(dbURL, "root", "root");
 } catch (Exception except) {
 except.printStackTrace();
 }
 return conn;
 }
}
```

## 5. Feladat

Alakítsuk át az alkalmazásunkat, hogy Connection objektum helyett a hatékonyabb DataSource objektumot használja. A DataSource paramétereit (url, driverClassName, username, password) vezessük ki property file-ba, a @Value annotáció használatával pedig property file-ból adjuk meg a paramétereket.

### Megoldás

Az InfraConfiguration osztályt alakítsuk át. Vegyük ki a Connection objektumot, helyette használjuk a DriverManagerDataSource objektumot.

```
@Configuration
public class InfraConfiguration {

 @Bean
 public DataSource dataSource() {
 DriverManagerDataSource ds = new DriverManagerDataSource();

 ds.setDriverClassName("org.mariadb.jdbc.Driver");
 ds.setUrl("jdbc:mariadb://localhost:3306/transfer");
 ds.setUsername("root");
 ds.setPassword("root");
 return ds;
 }
}
```

Alakítsuk át az ApplicationConfiguration és JdbcAccountRepository osztályokat, hogy Connection helyett DataSource objektumot használjanak.

Futtassuk le a teszt programot.

Hozzuk létre a db.properties file-t a com.masterfield.configuration alatt az alábbi tartalommal:

```
db.driver=org.mariadb.jdbc.Driver
db.url=jdbc:mariadb://localhost:3306/transfer
db.username=root
db.password=root
```

Adjuk meg a PropertySource annotációt az InfraConfiguration-nél.

```
@PropertySource("classpath:/hu/masterfield/bankproject/configuration/db.properties")
```

Módosítsuk a DataSource beállítást, hogy a @Value annotációkkal jelölt változók értékét használja.

```
@Bean
public DataSource dataSource(
 @Value("${db.driver}") String driverClassName,
 @Value("${db.url}") String url,
 @Value("${db.username}") String username,
 @Value("${db.password}") String password) {
 DriverManagerDataSource ds = new DriverManagerDataSource();

 ds.setDriverClassName(driverClassName);
 ds.setUrl(url);
 ds.setUsername(username);
 ds.setPassword(password);

 return ds;
}
```

## 6. Feladat

Módosítsuk banki alkalmazásunkat, hogy komponenseket használjon (@Component).

### Megoldás

A TransferServiceImpl osztályunkat lássuk el a @Service annotációval.

```
@Service("transferService")
```

Módosítsuk, hogy az AccountRepository esetén automatikusan a JdbcAccountRepository bean-t szűrje be.

```
@Autowired
public TransferServiceImpl(@Qualifier("JdbcAccountRepository")
 AccountRepository accountRep) {
 this.accountRep = accountRep;
}
```

Jelöljük @Component annotációval a JdbcAccountRepository osztályunkat is:

```
@Repository("JdbcAccountRepository")
```

Kössük be a DataSource változót.

Készítsük el a @PostConstructor és @PreDestroy életciklus függvényeket.

Módosítsuk az ApplicationConfiguration osztályt: vegyük ki a Bean és DataSource definíciókat.

Állítsuk be a komponensek keresését az alábbi annotációval:

```
@ComponentScan({"hu.masterfield.bankproject.services", "com.masterfield.bankproject.repo
sitory"})
```

## 7. Feladat

Módosítsuk banki alkalmazásunkat, hogy XML alapú konfigurációs file-t használjon a Spring bean-ek leírására.

### Megoldás

Hozzuk létre az application-config.xml file-t a hu.masterfield.bankproject.configuration alatt az alábbi tartalommal:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

 <context:property-placeholder

 location="classpath:hu/masterfield/bankproject/configuration/db.properties" />

 <bean id="transferService"
 class="hu.masterfield.bankproject.services.TransferServiceImpl">
 <property name="accountRepository" ref="accountRepository" />
 </bean>

 <bean id="accountRepository"
 class="hu.masterfield.bankproject.services.JdbcAccountRepository">
 <constructor-arg ref="dataSource"/>
 </bean>
</beans>
```

Hozzuk létre egy XMLApplicationConfiguration osztályt a com.masterfield.configuration alatt.

@ImportResource annotációval adjuk meg az application-config.xml file-t.

```
@Configuration
@ImportResource("classpath:hu/masterfield/bankproject/configuration/application-
config.xml")
@Import(InfraConfiguration.class)
public class XMLApplicationConfiguration {
}
```

Töröljük a @Component (@Service, @Repository) annotációt a JdbcAccountRepository és TransferServiceImpl osztályokról. Valamint nincs szükség a @Qualifier annotáció használatára sem.

Módosítsuk teszt osztályainkat, hogy az XMLApplicationConfiguration osztályt használják a Spring konfiguráció meghatározására.

```
ApplicationContext context = SpringApplication.run(XMLApplicationConfiguration.class);
TransferService service = (TransferService)context.getBean("transferService")
```

## 8. Feladat

a)

Állítsuk be az alábbi logolási szinteket az application.properties-ben:

```
logging.level.web=debug
logging.level.hu.masterfield=error
logging.level.org=error
```

Készítsünk futtatható jar csomagot a Spring Boot Maven plugin segítségével. BankProjectApplication osztályon szüntessük meg a @SpringBootApplication jelölést, helyette a TransferTestSpring osztályt jelöljük meg. (A feladat befejezésekor ezt állítsuk vissza.)

Futtassuk (F1) a „Maven: Execute Commands” parancsot és adjunk meg egy sajátot (Custom):  
package -DskipTests

Target könyvtárban megtaláljuk az elkészült file-okat.

Futtassuk a létrejött jar-t: c:\jdk-17.0.5+8\bin\java -jar .\bankproject-0.0.1-SNAPSHOT.jar

A webes projekt elkészülte után a feladat megismételhető, ekkor már a Tomcat szerver is elindul. Ehhez először módosítsuk a pom.xml esetén a csomagolást war-ra (<packaging>war</packaging>).

- b) Állítsunk be cache-t a JdbcAccountRepository loadAccount() függvényéhez, hogy az adatbázis lekérdezést csak egyszer végezze el és utána már a cache-ből adja vissza az Account-ot.

### Megoldás

Adjuk hozzá a pom.xml-hez a szükséges függőséget:

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-cache</artifactId>
 <version>3.0.1</version>
</dependency>
```

Adjuk hozzá a @SpringBootApplication és @EnableCaching annotációkat az XMLApplicationConfiguration-höz.

Jelöljük meg a JdbcAccountRepository loadAccount() függvényét:

```
@Cacheable(value = "accountCache", key = "#accountName")
```

Készítsünk egy CacheTest JUnit teszt osztályt.

## 9. Feladat

Készítsünk saját mock tesztet (MockTest.java), amely egy Subscriber interface-en végez teszteket. Használjuk az EasyMock eszközt.

### Megoldás

Adjuk hozzá az EasyMock könyvtárát a pom.xml Maven file-hoz:

```
<dependency>
 <groupId>org.easymock</groupId>
 <artifactId>easymock</artifactId>
 <version>3.4</version>
</dependency>
```

Hozzuk létre a hu.masterfield.bankproject.test csomag alatt a MockTest osztályt. Az osztályon belül deklaráljuk a Subscriber interface-t.

```
public class MockTest {

 interface Subscriber {
 public String getName();
 public int getAge();
 public int setBalance(int balance);
 }

 public static void main(String[] args) {
 Subscriber subs = createMock(Subscriber.class);

 expect(subs.getName()).andReturn("Kovács Szilárd");
 expect(subs.getAge()).andReturn(12);
 expect(subs.setBalance(100)).andReturn(777).times(1).andThrow(new
 RuntimeException("Exception occurred!"));
 replay(subs);

 System.out.println(subs.getName());
 System.out.println(subs.getAge());
 System.out.println(subs.setBalance(100));
 try {
 System.out.println(subs.setBalance(100));
 } catch (Exception e) {
 e.printStackTrace();
 }

 verify(subs);
 }
}
```

## 10. Feladat

- a) Készítsünk el egy `initbalance.sql`-t, amely a tesztelés során az `ACCOUNTS` adatbázis táblában alaphelyzetbe (10000) állítja az egyenlegeket. Futtassuk a `JUnit TransferServiceTests` teszt során az `initbalance.sql`-t.

### Megoldás

Adjunk hozzá a projekthez egy új SQL file-t a `hu.masterfield.bankproject` alá (a JUnit teszt osztályunk mellé) `initbalance.sql` néven az alábbi tartalommal:

```
UPDATE ACCOUNTS SET ACCOUNT_BALANCE=10000;
```

Módosítsuk a `TransferServiceTests` JUnit teszt osztályunkat.

Adjuk hozzá a következő annotációkat a tesztelés támogatásához:

```
@SpringBootTest
@ContextConfiguration(classes=XMLApplicationConfiguration.class)
```

Így a teszteseteink közös `ApplicationContext` objektumot fognak használni, azaz a Spring konténer csak egyszer példányosodik. Kössük be a `TransferService`-t `@Autowired` annotáció használatával.

A `transfer` metódusnál helyezzük el az alábbi annotációt:

```
@Sql(scripts="initbalance.sql",
executionPhase=Sql.ExecutionPhase.BEFORE_TEST_METHOD)
```

- b) Hozzunk létre egy tesztelési erőforrás file-t (`test_data.properties` néven). Adjunk meg tesztadatokat és használjuk fel őket az átutalás tesztelésénél.

### Megoldás

A `test/resources` könyvtárban hozzunk létre egy `test_data.properties` file-t. Adjuk meg az adatokat:

`test.fromAccountName=Szegény Sándor`

`test.toAccountName=Pénzes Zsolt`

`test.amount=900`

Állítsuk be az ISO 8859-1 kódolást a file-on: ISO 8859-1

Adjuk meg a `@TestPropertySource` annotációt a teszt osztályon:

```
@TestPropertySource("/test_data.properties")
```

Olvassuk be az adatokat a teszt metódus argumentumaiban:

```
public void transfer(@Value("${test.fromAccountName}") String fromAccountName,
 @Value("${test.toAccountName}") String toAccountName,
 @Value("${test.amount}") int amount) {
```

## 11. Feladat

Készítsünk egy MetricsAspect osztályt, amely futási időt mér a diákon megismert példa alapján a transfer() metódus esetén.

### Megoldás

Hozzunk létre egy új csomagot hu.masterfield.bankproject.aspects néven. Itt hozzuk létre a MetricsAspect osztályt az alábbi tartalommal:

```
@Aspect
@Component
public class MetricsAspect {

 @Around("execution(* transfer(..))")
 public Object profile(ProceedingJoinPoint point) throws Throwable {
 long time = System.currentTimeMillis();
 try {
 return point.proceed();
 } finally {
 long responseTime = System.currentTimeMillis() - time;
 System.out.println("<< " + point + " >> took " +
 responseTime + " msec");
 }
 }

 @AfterReturning(value="execution(* transfer(..))", returning="conf")
 public void dump(JoinPoint point, Confirmation conf) {
 System.out.println("<< " + point + " >> dump=" + conf);
 }
}
```

Engedélyezzük az aspektusok használatát az XMLApplicationConfiguration-ben:

```
@EnableAspectJAutoProxy
@ComponentScan("hu.masterfield.bankproject.aspects")
```

Futtassuk a TransferTestSpring teszt osztályunkat.  
Készítsünk az input adatokat is kidumpoló függvényt.



## 12. Feladat

Módosítsuk a JdbcAccountRepository-t, hogy JdbcTemplate-et használjon. Készítsünk dump...() függvényeket a diasoron látott példák kipróbálásához, melyek kiírják az adatbázis tábla tartalmát. A függvényeket a konstruktorból hívjuk meg.

Bővítsük a JUnit teszt osztályunkat: JdbcTestUtils osztály metódusainak használatával töröljük a teszt elején az ACCOUNTS tábla tartalmát, majd a transfer hívás után ellenőrizzük a létrejött rekordok számát.

## Megoldás

Módosítsuk a JdbcAccountRepository osztályt. Vegyük fel a JdbcTemplate attribútumot:

```
private JdbcTemplate jdbcTemplate = null;
```

Hozzuk létre a JdbcTemplate-et a konstruktorban a DataSource alapján:

```
public JdbcAccountRepository(DataSource dataSource) {
 this.jdbcTemplate = new JdbcTemplate(dataSource);
}
```

Írjuk át a loadAccount(), insertAccount(), updateAccount() függvényeket, hogy JdbcTemplate-et használjanak. Készítsük el a dump...() függvényeket.

Bővítsük a tesztet a JdbcTestUtils osztály metódusainak használatával:

```
JdbcTestUtils.deleteFromTables(jdbcRep.getJdbcTemplate(), "ACCOUNTS");
Assertions.assertEquals(0, JdbcTestUtils.countRowsInTable(jdbcRep.getJdbcTemplate(),
"ACCOUNTS"));
```

Szükséges a JdbcAccountRepository kiegészítése a getJdbcTemplate() függvénnyel és a JUnit-ba történő beillesztése @Autowired-del.

### 13. Feladat

- a) Készítsük el az alkalmazás JPA-Hibernate változatát. Hozzunk létre egy JpaAccountRepository-t és valósítsuk meg az adatbázis logikát entitások segítségével. Írjuk át JPA entitásra az Account adatszerkezetünket. Készítsünk dump() függvényt, amely az adatbázis tartalmát írja ki JPA-Hibernate segítségével. Első esetben JPA Criteria API, második esetben natív lekérdezés használatával.

#### Megoldás

Az application-config.xml-ben írjuk át, hogy a JpaAccountRepository-t használja az alkalmazás.

```
<bean id="accountRepository"
 class="hu.masterfield.bankproject.services.JpaAccountRepository">
</bean>
```

Kapcsoljuk be a tranzakciókezelést a Spring konténerben: adjuk hozzá a @EnableTransactionManagement annotációt az InfraConfiguration-höz.

És állítsuk be az entityManagerFactory-t és transactionManager-t:

```
@Bean
public JpaTransactionManager transactionManager(@Named("dataSource") DataSource dataSource) {
 JpaTransactionManager trxMan = new JpaTransactionManager();
 trxMan.setDataSource(dataSource);
 return trxMan;
}

@Bean(name = "entityManagerFactory")
public LocalContainerEntityManagerFactoryBean entityManagerFactory(@Named("dataSource")
DataSource dataSource) {
 LocalContainerEntityManagerFactoryBean emFactory = new
LocalContainerEntityManagerFactoryBean();
 emFactory.setDataSource(dataSource);
 emFactory.setPackagesToScan("hu.masterfield.bankproject.datatypes");

 HibernateJpaVendorAdapter jpaVendorAdapter = new HibernateJpaVendorAdapter();
 jpaVendorAdapter.setShowSql(true);
 jpaVendorAdapter.setGenerateDdl(false);
 jpaVendorAdapter.setDatabase(Database.MYSQL);

 emFactory.setJpaVendorAdapter(jpaVendorAdapter);
 Properties jpaProperties = new Properties();
 jpaProperties.setProperty("hibernate.format_sql", "true");

 emFactory.setJpaProperties(jpaProperties);
 return emFactory;
}
```

Módosítsuk az Account adatszerkezetünket. Annotációk segítségével kössük hozzá a megfelelő adatbázis elemekhez:

```
@Entity
@Table(name="ACCOUNTS")
public class Account implements Serializable {

 @Id
 @Column(name="ACCOUNT_NAME")
 private String accountName = null;

 @Column(name="ACCOUNT_BALANCE")
```

```
@Transient
private String password = null;
```

Készítsük el a JpaAccountRepository osztályt:

```
@Transactional
public class JpaAccountRepository implements AccountRepository {

 @PersistenceContext
 private EntityManager em;

 @Override
 public Account loadAccount(String accountName) throws SQLException {
 dumpAllAccounts();
 Account acc = em.find(Account.class, accountName);

 if (acc == null) {
 acc = new Account(accountName, 10000);
 insertAccount(acc);
 }

 return acc;
 }

 @Override
 public void updateAccount(Account account) throws SQLException {
 em.merge(account);
 }

 @Override
 public void insertAccount(Account account) throws SQLException {
 em.persist(account);
 }
}
```

Készítsük el a dump() függvényeket:

[illegible]

b) [Opcionális] JPA megoldásunkat alakítsuk át, hogy beágyazott H2 adatbázist használjon.

## Megoldás

Adjuk hozzá a szükséges library-t a pom.xml-hez:

```
<dependency>
 <groupId>com.h2database</groupId>
 <artifactId>h2</artifactId>
 <scope>runtime</scope>
</dependency>
```

Adjunk meg egy új db.embedded.properties file-t az alábbi beállításokkal:

```
db.driver=org.h2.Driver
db.url=jdbc:h2:mem:transfer;DB_CLOSE_DELAY=-1
db.username=root
db.password=root
```

Állítsuk át erre az InfraConfiguration-t:

```
@PropertySource("classpath:/hu/masterfield/bankproject/configuration/db.embedded.properties")
```

InfraConfiguration-ben adjuk meg a generálást és állítsuk be a H2 dialektust:

```
jpaVendorAdapter.setGenerateDdl(true);
jpaVendorAdapter.setDatabase(Database.H2);
```

- c) [Opcionális] Vegyünk fel egy új entitás osztályt *Contact* néven, amely tartalmazzon egy

**ContactType contactType:** a megadott kapcsolati típus, felsorolás típus az alábbiakkal:  
állandó lakcím, ideiglenes lakcím, telefon, email

**String contactData:** a megadott kapcsolati adat

A TransferService implementációs osztályban alakítsunk ki *addContacts()* és *getContacts()* függvényeket, amelyek hívásával egy megadott accounthoz rendelhetünk különböző kapcsolati adatokat.

#### 14. Feladat

Készítsünk Webes felületet a banki átutaláshoz. Az alkalmazás a <http://localhost:8080/> címen lesz elérhető. Egy form-on legyen lehetőség megadni a „From Account Name”, „To Account Name” és „Amount” mezőket. Majd egy TRANSFER gombra kattintva végezze el az átutalást a korábban létrehozott TransferService segítségével.

#### Megoldás

Készítsük el az index.html oldalt a beviteli formmal és helyezzük el a `src/main/resources/static/` folder alá.

Valósítsuk meg a TransferController.java feldolgozó osztály a `hu.masterfield.bankproject.controllers` csomag alatt:

```
@Controller
public class TransferController {
 private TransferService service;

 @Autowired
 public TransferController(TransferService service) {
 this.service = service;
 }

 @RequestMapping(value = "/transfer", method = {RequestMethod.POST})
 public String transfer(@RequestParam("fromAccountName") String fromAccountName,
 @RequestParam("toAccountName") String toAccountName,
 @RequestParam("amount") int amount,
 Model model,
 HttpServletRequest request,
 HttpSession session) {
 Confirmation receipt = null;
 receipt = service.transfer(fromAccountName, toAccountName, amount);
 System.out.println(receipt.getMessage());
 model.addAttribute("receipt", receipt);
 return "transferView";
 }
}
```

Készítsük el a nézetet transferView.html néven és helyezzük el a `src/main/resources/templates/` folder alá:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Money transfer receipt</title>
</head>
<body>
<p th:text="'New balance=' + ${receipt.newBalance}" />
<p th:text="'Success=' + ${receipt.success}" />
<p th:text="'Message=' + ${receipt.message}" />
</body>
</html>
```

Az application.properties file-ban állítsuk át servlet-re a web alkalmazás típusát:

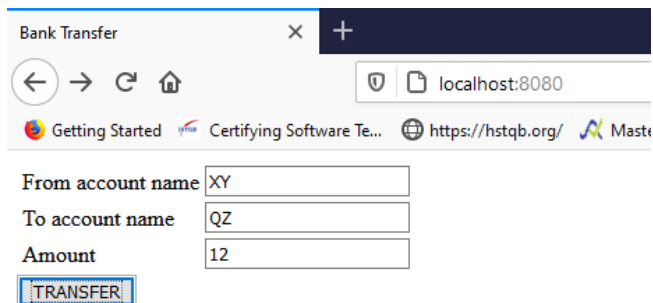
```
spring.main.web-application-type=servlet
```

Beállíthatunk magasabb logolást a webes konténerre:

```
logging.level.web=DEBUG
```

A BankprojectApplication osztályra tegyük vissza a @SpringBootApplication jelölést és futtassuk.

Próbáljuk ki az alkalmazást a <http://localhost:8080> címen.



Bank Transfer

localhost:8080

Getting Started Certifying Software Te... https://hstqb.org/ Master

From account name XY

To account name QZ

Amount 12

TRANSFER

Masterfield Oktatóközpont  
Spring programozás tanfolyam  
2024. november 13.  
Félegyházi Dávid

## 15. Feladat

Az alábbi feladatokhoz készítsük el a kliens oldalt is RestTemplate használatával. A transfer metódus(ok)hoz hozzunk létre MockMvc teszteseteket is. A c) részfeladathoz készítsünk teljes mock MVC tesztet.

a)

Készítsünk REST webservice-t, amely a transfer függvényt hívja meg.

A webservice meghívás az alábbi URL-lel történik:

<http://localhost:8080/rest?fromAccountName=AAA&toAccountName=BBB&amount=33>

b)

Alakítsuk át JSON POST feldolgozásra. Készítsünk teszt input file-t és a *curl* program használatával teszteljük le a függvényt.

c)

Készítsük el a datetime függvényt. Az alkalmazás URL-jében adjuk meg, hogy a dátumot vagy az időt szeretnénk lekérdezni:

Dátum:

<http://localhost:8080/rest/datetime/date>

Idő:

<http://localhost:8080/rest/datetime/time>

d)

Készítsünk egy *transferBatch()* REST webservice metódust, amely JSON formátumban kap meg több átutalást egyszerre, tömb formába és elvégzi azokat. A nyugtákat pedig egy listában gyűjtve adja vissza tömbként JSON formátumban.

e)

Módosítsuk az a) feladatot, hogy a nyugtát XML formában adja vissza a szolgáltatás. Ehhez egy új metódust ajánljunk ki */rest/xml* címen.

f) [Opcionális]

Készítsünk REST metódusokat az alábbi feladatokhoz:

- Adott Contact rekord lekérdezése ID alapján: az ID URL paraméterként kerüljön átadásra.
- Egy megadott account-hoz tartozó Contact rekordok lekérdezése
- Teljes Account és Contact táblák tartalmának visszaadása JSON formátumban

## Megoldás

a)

Készítsük el a TransferRestController REST WS-t megvalósító osztályt:

```
@RestController
public class TransferRestController {

 @Autowired
 private TransferService service;

 @GetMapping("/rest")
 public Confirmation transfer(
 @RequestParam(value="fromAccountName") String fromAccountName,
 @RequestParam(value="toAccountName") String toAccountName,
 @RequestParam(value="amount") int amount) {
 Confirmation receipt = null;
 receipt = service.transfer(fromAccountName, toAccountName, amount);
 System.out.println(receipt.getMessage());
 return receipt;
 }
}
```

b)

Adjunk hozzá egy új metódust a TransferRestController-hez, amely JSON formátumú input és output adatokat használ.

```
@PostMapping(path = "/rest",
 consumes = MediaType.APPLICATION_JSON,
 produces = MediaType.APPLICATION_JSON)
public Confirmation transfer(@RequestBody Map<String, String>
 payload) {
 String fromAccountName = (String)payload.get("fromAccountName");
 String toAccountName = (String)payload.get("toAccountName");
 int amount = Integer.parseInt(payload.get("amount"));
 Confirmation receipt = null;
 receipt = service.transfer(fromAccountName, toAccountName, amount);
 System.out.println(receipt.getMessage());
 return receipt;
}
```

Teszteljük az alkalmazást a curl használatával. Futtassuk a test1.cmd parancsot.

```
curl -L -v -i -H "Content-Type:application/json" -X POST
 http://localhost:8080/rest -d @body.json
```

c)

TransferRestController osztályban hozzuk létre a getDateOrTime függvényt:

```
@GetMapping("/datetime/{dateortime}")
public String getDateOrTime(@PathVariable String dateortime) {
 if(dateortime.equals("date")) {
 DateTimeFormatter dtf =
 DateTimeFormatter.ofPattern("yyyy.MM.dd");
 LocalDate localDate = LocalDate.now();
 return dtf.format(localDate);
 }
}
```



```
 if(dateortime.equals("time")) {
 DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("HH:mm:ss");
 LocalDateTime now = LocalDateTime.now();
 return dtf.format(now);
 }

 return "N/A";
 }
}
```

Teszteljük az alkalmazást a fenti URL-ek kipróbálásával.

Masterfield Oktatóközpont  
Spring programozás tanfolyam  
2024. november 13.  
Félegyházi Dávid

### **Ajánlott linkek, referenciák, tutorialok:**

- Maven Repository:  
<https://mvnrepository.com/>
- Spring Boot – Getting Started  
<https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html>
- Spring Boot – „How-to” Guides  
<https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html>
- Using Spring Boot  
<https://docs.spring.io/spring-boot/docs/current/reference/html/using-spring-boot.html>
- Spring Boot Without A Web Server  
<https://www.baeldung.com/spring-boot-no-web-server>
- Serving Web Content with Spring MVC  
<https://spring.io/guides/gs/serving-web-content/>
- Simplest Spring MVC Framework Tutorial – Hello World Example with UI (JSP) Page  
<https://crunchify.com/simplest-spring-mvc-hello-world-example-tutorial-spring-model-view-controller-tips/>
- Spring boot: Path with “WEB-INF” or “META-INF”  
<https://www.codersdesks.com/spring-boot-path-with-web-inf-or-meta-inf/>
- A Guide to the ViewResolver in Spring MVC  
<https://www.baeldung.com/spring-mvc-view-resolver-tutorial>
- Tutorial: Thymeleaf + Spring  
<https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>
- Spring - MVC Framework  
[https://www.tutorialspoint.com/spring/spring\\_web\\_mvc\\_framework.htm](https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm)
- Enable/disable Tomcat server  
<https://stackoverflow.com/questions/32078015/spring-boot-enable-disable-embedded-tomcat-with-profile>
- Embedded Web Servers  
<https://docs.spring.io/spring-boot/docs/2.1.9.RELEASE/reference/html/howto-embedded-web-servers.html>
- Serve Static Resources with Spring  
<https://www.baeldung.com/spring-mvc-static-resources>
- Spring Boot JSP View Resolver Example  
<https://howtodoinjava.com/spring-boot/spring-boot-jsp-view-example/>
- Testing  
<https://docs.spring.io/spring-framework/docs/current/reference/html/testing.html>
- JUnit 5 Tutorial: Writing Assertions With JUnit 5 Assertion API  
<https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-assertions-with-junit-5-api/>
- Testing the Web Layer  
<https://spring.io/guides/gs/testing-web/>
- Testing MVC Web Controllers with Spring Boot and @WebMvcTest  
<https://reflectoring.io/spring-boot-web-controller-test/>
- The BeanDefinitionOverrideException in Spring Boot  
<https://www.baeldung.com/spring-boot-bean-definition-override-exception>
- Unable to Locate Spring NamespaceHandler for XML Schema Namespace  
<https://www.baeldung.com/unable-to-locate-spring-namespacehandler-for-xml-schema-namespace>